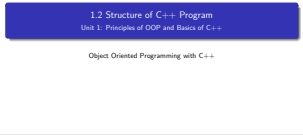
1.2 Structure of C++ Program

Unit 1: Principles of OOP and Basics of C++

Object Oriented Programming with C++



Lecture 6: Structure of a C++ Program & Fundamentals

- Unit 1: Principles of OOP and Basics of C++
- Topic 1.2: Structure of C++ Program
- Key elements: Tokens, Keywords, Identifiers, Variables, Data Types, Reference Variables

Lecture 6: Structure of a C++ Program & Fundamentals
- Unit 1: Principles of OOP and Basics of C++
- Topic 1.2: Structure of C++ Program
- Key elements: Tokens, Keywords, Identifiers, Variables, Data Types, Reference Variables

Welcome everyone to Lecture 6. Today we transition from high-level OOP concepts down to the actual syntactical building blocks of C++. Just as English has letters, words, and grammar, C++ has tokens, variables, and syntax rules. By the end of this lecture, you will know exactly how to structure a basic C++ program and how to store data using various types.

1. Structure of a C++ Program

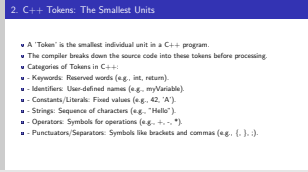
- Every C++ program has a specific layout or skeleton.
- 1. Documentation Section: Comments describing the program.
- 2. Preprocessor Directives: E.g., #include ¡iostream¿ to include standard libraries.
- 3. Global Declarations: Variables or functions accessible throughout the program.
- 4. main() Function: The entry point of every C++ program.
- 5. Subprograms: User-defined functions (often defined after main).

1. Structure of a C++ Program
- Every C++ program has a specific layout or skeleton.
- 1. Documentation Section: Comments describing the program.
- 2. Preprocessor Directives: E.g., #include ¡iostream¿ to include standard libraries.
- 3. Global Declarations: Variables or functions accessible throughout the program.
- 4. main() Function: The entry point of every C++ program.
- 5. Subprograms: User-defined functions (often defined after main).

Let's look at the anatomy of a C++ program. When the compiler looks at your code, it expects a certain order. While C++ is flexible, a standard structure includes documentation (comments) at the top, followed by preprocessor directives like #include which tell the compiler to fetch library code. Then comes the most critical part: the main() function. The execution of every C++ program begins at main(). Without it, your program will not run.

2. C++ Tokens: The Smallest Units

- A 'Token' is the smallest individual unit in a C++ program.
- The compiler breaks down the source code into these tokens before processing.
- Categories of Tokens in C++:
- - Keywords: Reserved words (e.g., int, return).
- - Identifiers: User-defined names (e.g., myVariable).
- - Constants/Literals: Fixed values (e.g., 42, 'A').
- - Strings: Sequence of characters (e.g., "Hello").
- - Operators: Symbols for operations (e.g., +, -, *).
- - Punctuators/Separators: Symbols like brackets and commas (e.g., {, }, ;).



Think of tokens as the atomic particles of a C++ program. When you write a line of code like 'int x = 10;', the compiler reads this as a sequence of tokens: 'int' is a keyword, 'x' is an identifier, '=' is an operator, '10' is a constant, and ';' is a punctuator. Understanding tokens helps you understand compiler errors, particularly syntax errors which often state 'unexpected token'.

3. Keywords in C++

- Keywords are reserved words that have a special, predefined meaning to the C++ compiler.
- They cannot be used as names for variables, classes, or functions.
- Always written in lowercase.
- Common Examples:
- - Data types: int, float, char, bool, void, double.
- - Control flow: if, else, switch, for, while, do, break, continue.
- - OOP concepts: class, public, private, protected, virtual, this.

3. Keywords in C++
- Keywords are reserved words that have a special, predefined meaning to the C++ compiler.
- They cannot be used as names for variables, classes, or functions.
- Always written in lowercase.
- Common Examples:
- - Data types: int, float, char, bool, void, double.
- - Control flow: if, else, switch, for, while, do, break, continue.
- - OOP concepts: class, public, private, protected, virtual, this.

Keywords are the vocabulary of the C++ language itself. Because the compiler uses them to understand the structure and commands of your program, you are forbidden from using them for your own variable names. For example, you cannot name a variable 'class' or 'int'. C++ has around 95 reserved keywords, though you will only regularly use a subset of them.

4. Identifiers: Naming Your Entities

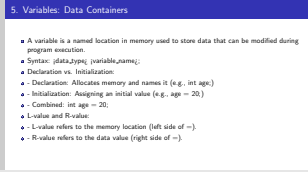
- Identifiers are the names assigned by the programmer to variables, functions, arrays, classes, etc.
- Rules for Naming Identifiers:
- 1. Can contain alphabets (A-Z, a-z), digits (0-9), and underscores (_).
- 2. Must begin with a letter or an underscore, NOT a digit.
- 3. Cannot be a C++ keyword.
- 4. C++ is case-sensitive (e.g., 'count' is different from 'Count').
- Valid: age, _temp, student1, calculateTotal
- Invalid: 1stStudent (starts with digit), int (keyword), my name (contains space)

4. Identifiers: Naming Your Entities
- Identifiers are the names assigned by the programmer to variables, functions, arrays, classes, etc.
- Rules for Naming Identifiers:
- 1. Can contain alphabets (A-Z, a-z), digits (0-9), and underscores (_).
- 2. Must begin with a letter or an underscore, NOT a digit.
- 3. Cannot be a C++ keyword.
- 4. C++ is case-sensitive (e.g., 'count' is different from 'Count').
- Valid: age, _temp, student1, calculateTotal
- Invalid: 1stStudent (starts with digit), int (keyword), my name (contains space)

While keywords belong to C++, identifiers belong to you, the programmer. You use them to name your data and functions. But you must follow the rules. A common mistake by beginners is starting a variable name with a number, or putting a space in it. Remember, C++ is strictly case-sensitive. 'Apple' and 'apple' are two completely different identifiers. It is best practice to use meaningful names like 'employeeSalary' rather than just 'x' or 'y'.

5. Variables: Data Containers

- A variable is a named location in memory used to store data that can be modified during program execution.
- Syntax: ¡data_type¿ ¡variable_name¿;
- Declaration vs. Initialization:
- - Declaration: Allocates memory and names it (e.g., int age;)
- - Initialization: Assigning an initial value (e.g., age = 20;)
- - Combined: int age = 20;
- L-value and R-value:
- - L-value refers to the memory location (left side of =).
- - R-value refers to the data value (right side of =).



A variable is essentially a labeled box in the computer's RAM. When you say 'int age = 20;', you are telling the computer: 'Find a box, label it age, make sure it only holds integers, and put the number 20 inside it'. It's important to differentiate between declaring a variable (creating the box) and initializing it (putting the first item in the box). If you don't initialize a variable, it may contain 'garbage' data left over in that memory space.

6. Introduction to Data Types

- Data types define the type of data a variable can hold and the amount of memory it requires.
- They also determine which operations can be performed on the data.
- Categories of Data Types in C++:
- 1. Primitive / Built-in Types (int, char, float, double, bool, void)
- 2. Derived Types (Arrays, Pointers, Functions, References)
- 3. User-Defined Types (class, struct, union, enum)

6. Introduction to Data Types
- Data types define the type of data a variable can hold and the amount of memory it requires.
- They also determine which operations can be performed on the data.
- Categories of Data Types in C++:
- 1. Primitive / Built-in Types (int, char, float, double, bool, void)
- 2. Derived Types (Arrays, Pointers, Functions, References)
- 3. User-Defined Types (class, struct, union, enum)

Data types are crucial because memory is finite. The compiler needs to know exactly how much RAM to reserve for a variable and how to interpret the binary 1s and 0s stored there. For example, adding two integers is structurally different at the processor level than adding two floating-point decimals. Today, we will focus specifically on the Primitive or Built-in data types.

7. Primitive Data Types: Integers (int)

- Used to store whole numbers without fractional parts.
- Type: int
- Size: Typically 4 bytes (32 bits) on modern systems.
- Range: -2,147,483,648 to 2,147,483,647
- Modifiers can change size and range:
- - short int (usually 2 bytes)
- - long int (usually 4 or 8 bytes)
- - unsigned int (only positive numbers, doubles the positive range)

7. Primitive Data Types: Integers (int)
- Used to store whole numbers without fractional parts.
- Type: int
- Size: Typically 4 bytes (32 bits) on modern systems.
- Range: -2,147,483,648 to 2,147,483,647
- Modifiers can change size and range:
- - short int (usually 2 bytes)
- - long int (usually 4 or 8 bytes)
- - unsigned int (only positive numbers, doubles the positive range)

The 'int' is your go-to data type for counting things or representing whole numbers, like the number of students in a class. We also have modifiers. If you know a number will never be negative—like an age or a distance—you can use 'unsigned int'. This reclaims the bit used for the negative sign and doubles your maximum possible positive value.

8. Primitive Data Types: Floating Point (float, double)

- Used to store numbers with decimal or fractional parts.
- Type: float (Single precision)
- - Size: 4 bytes
- - Precision: ~7 decimal digits
- - Example: float pi = 3.14159f;
- Type: double (Double precision)
- - Size: 8 bytes
- - Precision: ~15 decimal digits
- - Example: double exactPi = 3.141592653589793;

8. Primitive Data Types: Floating Point (float, double)
- Used to store numbers with decimal or fractional parts.
- Type: float (Single precision)
- - Size: 4 bytes
- - Precision: ~7 decimal digits
- - Example: float pi = 3.14159f;
- Type: double (Double precision)
- - Size: 8 bytes
- - Precision: ~15 decimal digits
- - Example: double exactPi = 3.141592653589793;

When dealing with continuous measurements like weight, temperature, or currency, integers won't work. You need floating-point types. 'float' gives you 4 bytes of precision, but for scientific calculations or high-precision financial software, 'double' is preferred because it allocates 8 bytes, drastically reducing rounding errors. Notice the 'f' suffix used when assigning a literal to a float; without it, C++ treats decimal literals as doubles by default.

9. Primitive Data Types: Characters (char)

- Used to store a single character (letter, digit, or symbol).
- Type: char
- Size: 1 byte (8 bits).
- Syntax: Enclosed in single quotes (e.g., char grade = 'A';)
- Under the hood: Characters are stored as integers using the ASCII encoding standard.
- Example: 'A' is stored as 65, 'a' is stored as 97.

9. Primitive Data Types: Characters (char)
- Used to store a single character (letter, digit, or symbol).
- Type: char
- Size: 1 byte (8 bits).
- Syntax: Enclosed in single quotes (e.g., char grade = 'A';)
- Under the hood: Characters are stored as integers using the ASCII encoding standard.
- Example: 'A' is stored as 65, 'a' is stored as 97.

A 'char' stores a single character and requires only 1 byte of memory. It's vital to remember that chars use SINGLE quotes, unlike strings which use double quotes. Interestingly, C++ doesn't actually store the letter 'A' in memory. It stores the number 65, which corresponds to 'A' in the ASCII chart. This means you can actually perform arithmetic on characters, like adding 1 to 'A' to get 'B'.

10. Primitive Data Types: Boolean (bool) and Void

- Boolean (bool):
- - Used to represent logical values.
- - Can hold only two values: true or false.
- - Size: 1 byte.
- - Under the hood: true is typically 1, false is 0.
- Void (void):
- - Represents the absence of type.
- - Cannot be used to declare standard variables.
- - Commonly used for functions that do not return a value.

10. Primitive Data Types: Boolean (bool) and Void
- Boolean (bool):
- - Used to represent logical values.
- - Can hold only two values: true or false.
- - Size: 1 byte.
- - Under the hood: true is typically 1, false is 0.
- Void (void):
- - Represents the absence of type.
- - Cannot be used to declare standard variables.
- - Commonly used for functions that do not return a value.

The bool type is the foundation of logic in programming. It's used in conditions and loops to make decisions. While it technically only needs 1 bit, it occupies 1 byte in memory due to how RAM is addressed. The 'void' type is slightly abstract; it literally means 'nothing'. You will see 'void' constantly when we start writing functions that execute actions but don't need to hand a value back to the caller.

11. Reference Variables: Introduction

- A reference variable is an 'alias' or alternative name for an already existing variable.
- Syntax: ¡data_type¿ &¡reference_name¿ = ¡existing_variable¿;
- Example:
- int originalScore = 100;
- int &refScore = originalScore;
- Behavior:
- - Both originalScore and refScore refer to the exact same memory location.
- - Changing refScore directly changes originalScore.

11. Reference Variables: Introduction
- A reference variable is an 'alias' or alternative name for an already existing variable.
- Syntax: ¡data_type¿ &¡reference_name¿ = ¡existing_variable¿;
- Example:
- int originalScore = 100;
- int &refScore = originalScore;
- Behavior:
- - Both originalScore and refScore refer to the exact same memory location.
- - Changing refScore directly changes originalScore.

Now we move to a feature unique to C++ (not found in standard C): Reference Variables. Think of a reference as a nickname. If someone's name is Robert, and his nickname is Bob, they are the same person. If Bob gets a haircut, Robert gets a haircut. Similarly, 'refScore' is just another name for the memory location of 'originalScore'. They are inextricably linked.

12. Rules and Use Cases for Reference Variables

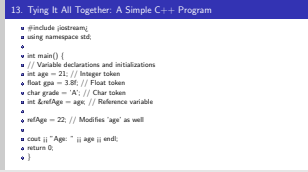
- Rules for References:
- 1. A reference MUST be initialized when it is declared.
- (e.g., int &ref; // ERROR)
- 2. Once bound to a variable, a reference cannot be re-bound to another variable.
- 3. You cannot have a reference to a null value.
- Primary Use Cases:
- - Pass-by-reference in functions: Allows a function to modify original arguments without copying data.
- - Creating aliases for complex, long variable names to improve readability.

12. Rules and Use Cases for Reference Variables
- Rules for References:
- 1. A reference MUST be initialized when it is declared.
- (e.g., int &ref; // ERROR)
- 2. Once bound to a variable, a reference cannot be re-bound to another variable.
- 3. You cannot have a reference to a null value.
- Primary Use Cases:
- - Pass-by-reference in functions: Allows a function to modify original arguments without copying data.
- - Creating aliases for complex, long variable names to improve readability.

References are strict. You cannot create a nickname for nothing; it must point to an existing variable immediately upon creation. Also, references are loyal; once bound to 'originalScore', 'refScore' cannot be reassigned to point to a different variable later. Their primary power lies in functions. By passing a reference to a function, you avoid making copies of large data structures, saving memory and processing time.

13. Tying It All Together: A Simple C++ Program

- #include ¡iostream¿
- using namespace std;
-
- int main() {
- // Variable declarations and initializations
- int age = 21; // Integer token
- float gpa = 3.8f; // Float token
- char grade = 'A'; // Char token
- int &refAge = age; // Reference variable
-
- refAge = 22; // Modifies 'age' as well
-
- cout ¡¡ "Age: " ¡¡ age ¡¡ endl;
- return 0;
- }



Let's put everything together in a complete program structure. We have our preprocessor directive at the top. We enter the main function. We declare identifiers like 'age', 'gpa', and 'grade' using keywords 'int', 'float', and 'char'. We create a reference variable 'refAge' tied to 'age'. Notice how modifying 'refAge' to 22 will result in the cout statement printing 22. This demonstrates how tokens, data types, variables, and references all interact in a real C++ environment.

14. Summary & Q&A

- Today we covered:
- - The mandatory structure of a C++ program (main function).
- - Tokens: Keywords and rules for Identifiers.
- - Variables and memory allocation.
- - Primitive Data Types: int, float, double, char, bool, void.
- - Reference Variables: Aliases and their strict rules.

14. Summary & Q&A
- Today we covered:
- - The mandatory structure of a C++ program (main function).
- - Tokens: Keywords and rules for Identifiers.
- - Variables and memory allocation.
- - Primitive Data Types: int, float, double, char, bool, void.
- - Reference Variables: Aliases and their strict rules.

To summarize, you now know how to frame a C++ program, how to legally name your variables, how to choose the right data type for your data to optimize memory, and how to create aliases using references. Are there any questions on the sizes of these data types, or how references differ from standard variables? In the lab session, we will write programs using all of these types.